RIKE 2015 Award 1344 – Enabling widespread electronic corrosion monitoring in marine off-shore installations – Project Report Part 2

Table of Contents

Table of Figures

Introduction

This report forms part of the documentation towards the Southampton Solent University 2015 RIKE award no. 1344 “Enabling widespread corrosion monitoring in marine off-shore installations”. It is a continuation of a previous document detailing corrosion detection technology at the time of writing.

This report documents the theoretical and practical results of the investigation into detecting rust using its non-conductive properties.

A simplified, magnified technical diagram of a rusty steel surface would look like Figure 1:

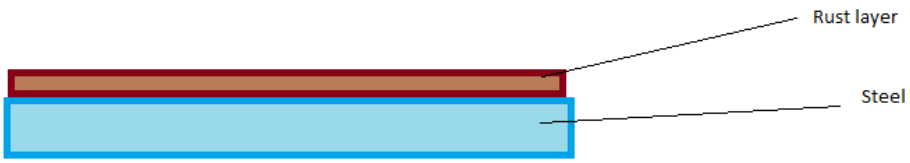


Figure 1: Simplified functional diagram of surface rust

The underlying steel layer is conductive to electricity, whereas the rust layer is non-conductive. The structure shown in Figure 1 can also be seen as part of an electrical capacitor, which is an electronic component consisting of two conductive layers separated by an insulator (1). In order to turn the structure in Figure 1 into a capacitor, a second conductive layer is needed, so that charge can be held between the two conductive layers. Adding this layer produces the structure seen in Figure 2.

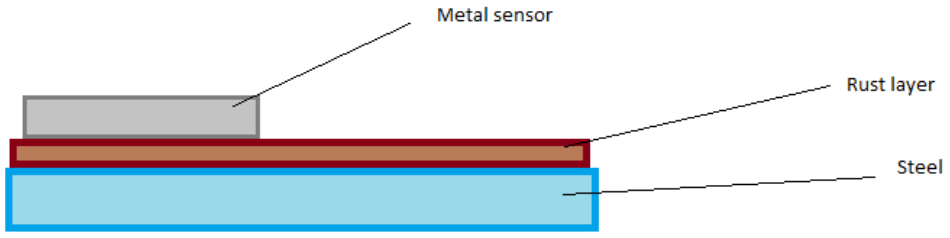


Figure 2: Capacitor structure created by adding a metal layer

The capacitance (a value to indicate the capacitor’s ability to hold charge) of a capacitor is given by:

$$C = \frac{\varepsilon_0 \varepsilon_r A}{d}$$

Where C = Capacitance, ε_0 = permittivity of free space, ε_r = relative permittivity of insulating material, also known as dielectric constant, A = area of commonality and d = thickness of insulating material.

If it were possible to link this capacitor to an electronic circuit that could measure capacitance, or that showed a change in output depending on change in capacitance, then Figure 2 could be seen to show a sensor that measured capacitance of rust. One such circuit is known as a 555 timer circuit. This makes use of a 555 timer integrated circuit which is used extensively in electronics for many

purposes. The 555 timer can be used in bistable, monostable and astable modes, but most often in the astable. The circuit diagram for an astable mode 555 timer is shown in Figure 3.

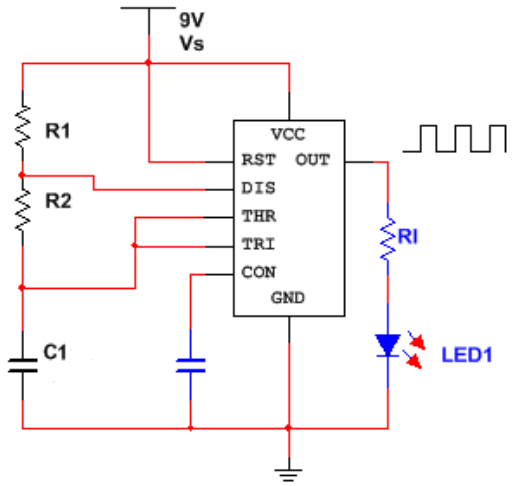


Figure 3: Circuit diagram for 555 timer in astable mode

In this circuit, when power is applied, the output from pin 3 is a square wave signal with a frequency dependent on the values of the resistors R_1 and R_2 and the capacitor C_1. The relationship between the component values and the frequency produced is given by:

$$f = \frac{1}{\ln(2).C_1.(R_1 + 2R_2)}$$

It can be seen from the above equation that the output frequency is partially dependent on the value of the capacitor in the circuit. If this capacitor is replaced by the structure shown in Figure 2, then a rust sensing device would be created, that exhibits a change in output frequency depending on the dielectric properties of the rust that forms the capacitor C_1. A 555 timer circuit in astable mode was built. The circuit diagram shown in Figure 4 has been drawn in MultiSim™:

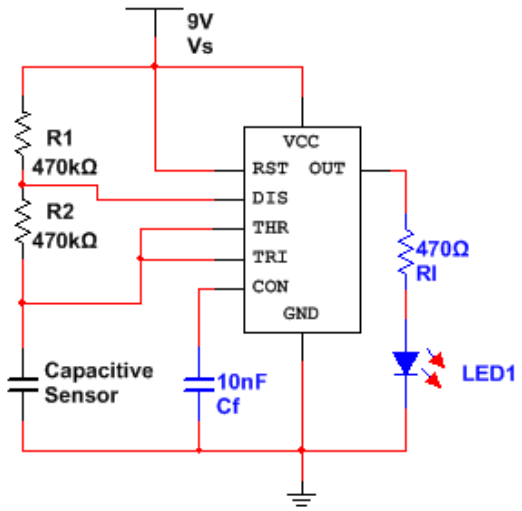


Figure 4: Circuit diagram of corrosion sensor circuit

R_1 and R_2 were chosen to be 470kΩ in order to make the frequency as low as possible. The capacitive sensor, shown in the lower left of the diagram, represents the corrosion sensor, and the connection to ground is realised by having a crocodile clip-terminated flying lead connection from the negative terminal of the battery, via the circuit board, to the steel structure underlying the rust layer.

Sensors

Various different sensor configurations have been created and used to try to detect rust.

Capacitance of copper tape sensor

The first sensor was made using layers of copper tape soldered to a wire which connected into the astable 555 timer circuit, shown in Figure 5.

Figure 5: Sensor made from copper tape

The area of the sensor shown in Figure 5 is approximately 23mm x 11mm. The shape is only approximately rectangular, so the area is estimated to be:

$$A \approx 23 \times 10^{-3} \times 11 \times 10^{-3}$$
$$= 2.53 \times 10^{-4} m^2$$

Capacitance is given by the following equation:

$$C = \frac{\varepsilon_0 \varepsilon_r A}{d}$$

Where C = Capacitance, ε_0 = permittivity of free space, ε_r = relative permittivity of insulating material, also known as dielectric constant, A = area of commonality and d = thickness of insulating material.

According to (2), the value of ε_r for rust is 13.1. The thickness of rust on our sample is estimated to be 1µm (from (3)).

For the copper tape sensor seen in Figure 5, the capacitance resulting in it being placed on a patch of rust of thickness 1µm can be calculated as:

$$C = \frac{\varepsilon_0 \varepsilon_r A}{d}$$

$$= \frac{8.854 \times 10^{-12} \times 13.1 \times 2.53 \times 10^{-4}}{1 \times 10^{-6}}$$

$$= 2.9345 \times 10^{-8} F$$

$$= 29.34\ nF$$

Results with copper tape sensor

Initial tests using the sensor shown in Figure 5 were encouraging, but it was found that the frequency output from the 555 timer circuit (and therefore the capacitance of the rust-dielectric capacitor) varied hugely depending on the pressure that was placed on the sensor. This is possibly because the rust is not flat but is spiky in cross-section, leaving a large air gap between the rusty surface and the sensor. When the sensor was pressed down, the air gap was reduced, but the rust was also crushed, and the senor was deformed meaning that results were not repeatable. It was decided that a new sensor was needed that was made of a material that was rigid rather than flexible.

FR4 Sensor

As a result, a new sensor was made using FR4 circuit board. This is a rigid material coated on one side with copper, used mostly to create printed circuit boards by etching away the copper layer selectively to leave copper tracks that can then be populated with electronic components to create a circuit board that performs a particular function. The FR4 polymer is rigid, the thickness is known and the dielectric constant of FR4 is known to be 4.5 at high frequencies (4) so the capacitance of a sensor made of FR4 can be calculated.

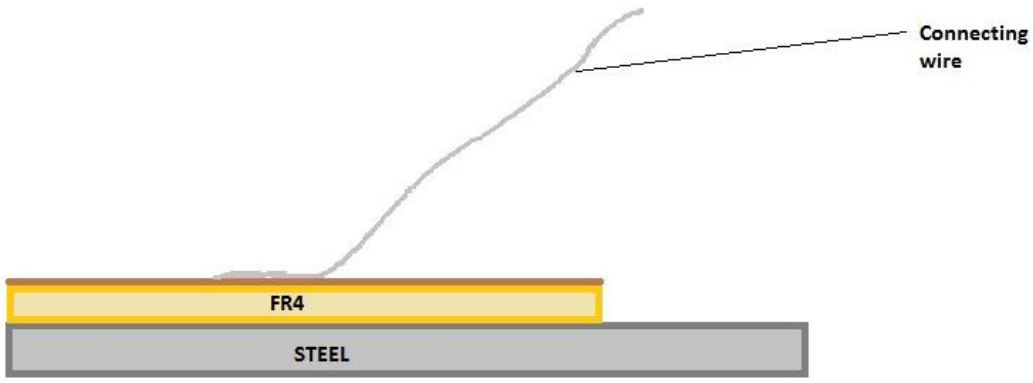


Figure 6: FR4 Sensor on top of steel

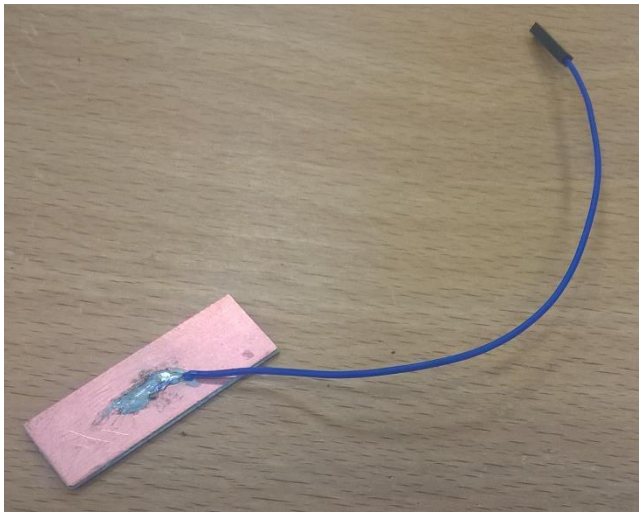


Figure 7: Practical realisation of FR4 sensor

Capacitance of FR4 sensor

Such a sensor was constructed and a photograph can be seen in Figure 7. The thickness of the FR4 is 1.4mm and the area is 42mm x 15mm.

If a known area A of FR4 is placed on top of metal, as in Figure 6, the capacitance can be calculated as follows:

$$\begin{aligned} C_{FR4} &= \epsilon_0 \epsilon_{FR4} \frac{A}{d} \\ &= \frac{8.854 \times 10^{-12} \times 4.5 \times 42 \times 10^{-3} \times 15 \times 10^{-3}}{1.4 \times 10^{-3}} \\ &= 1.7929 \times 10^{-11} F \\ &= 17.93 pF \end{aligned}$$

If the sensor is placed on an area of rust, in order to calculate the capacitance of rust in contact with the FR4 sensor described above, the value of the dielectric constant for rust needs to be known. The area of the capacitor is the same as above. To calculate the capacitance:

$$\begin{aligned} C_{rust} &= \frac{\epsilon_0 \epsilon_{rust} A}{d} \\ &= \frac{8.854 \times 10^{-12} \times 13.1 \times 42 \times 10^{-3} \times 15 \times 10^{-3}}{1 \times 10^{-6}} \\ &= 7.307 \times 10^{-8} F \\ &= 73.1\ nF \end{aligned}$$

Addition of these two capacitors in series, which is how they would appear electrically in a circuit, gives a total capacitance of:

$$\begin{aligned} \frac{1}{C_{total}} &= \frac{1}{C_{rust}} + \frac{1}{C_{sensor}} \\ &= \frac{1}{7.307 \times 10^{-8}} + \frac{1}{1.793 \times 10^{-11}} \\ \therefore C_{total} &= 1.792 \times 10^{-11} F \\ &= 17.92 pF \end{aligned}$$

This value is almost exactly the same as the sensor capacitance alone. That means that, if a 555 timer circuit were built with the sensor as the capacitor, the frequency of the output signal when the

sensor was touching rust would be almost identical to the frequency when it was touching no rust. This implies that a different sensor design is needed.

The thinnest FR4 that is commercially available is 0.3mm, used for high frequency antennas and circuitry. The capacitance of a capacitor made from this material would be:

$$\begin{aligned} C_{FR4} &= \epsilon_0 \epsilon_{FR4} \frac{A}{d} \\ &= \frac{8.854 \times 10^{-12} \times 4.5 \times 42 \times 10^{-3} \times 15 \times 10^{-3}}{0.3 \times 10^{-3}} \\ &= 8.367 \times 10^{-11} F \\ &= 83.67 pF \end{aligned}$$

This value, while larger than the first FR4 sensor, is still of the order of 1000 times smaller than the calculated capacitance for the area of rust that it covers, so a different design was sought.

Kapton Tape Sensor

If Kapton™ tape, a tape used in 3D-printers, is used, it has a $\varepsilon_{r\ of}$ 3.5 and a typical thickness of 13µm (although two adhered layers were measured by micrometer to have a thickness of 0.12mm). Using a thickness value of 13µm, the capacitance of a sensor created using Kapton tape with the same area would be:

$$\begin{aligned} C &= \frac{\epsilon_0 \epsilon_r A}{d} \\ &= \frac{8.854 \times 10^{-12} \times 3.5 \times 6.3 \times 10^{-4}}{13 \times 10^{-6}} \\ &= 1.5 \times 10^{-9} F \\ &= 1.5 nF \end{aligned}$$

This value is in the order of one fiftieth of the capacitance of the same area of 1µm thick rust. If these two sensors were connected in series, as would be the case if the circuit were viewed electrically when the Kapton sensor were placed on top of rust, the total capacitance can be calculated from:

$$\begin{aligned} C_{total} &= \frac{1}{\frac{1}{C_{rust}} + \frac{1}{C_{Kapton}}} \\ &= \frac{1}{\frac{1}{7.307 \times 10^{-8}} + \frac{1}{1.5 \times 10^{-9}}} \\ &= 1.4698 \times 10^{-9} F \\ &= 1.47 nF \end{aligned}$$

The difference between this value and the capacitance of the sensor touching rust-free metal is approximately 0.03nF, so there would be a change of approximately 2% between the sensor touching rust and touching metal.

Practical realization of Kapton Sensor

A sensor was created using Kapton tape as shown in Figure 8. It was envisaged that the Kapton sensor would be connected to the 555 timer circuit and would measure rust-free metal on one side and rusty surface corrosion on the other. This configuration was utilized because of the variation discovered when the copper tape sensor was pressed down onto the rust, which was put down to the presence of an air gap. The measurement configuration can be seen in Figure 9.

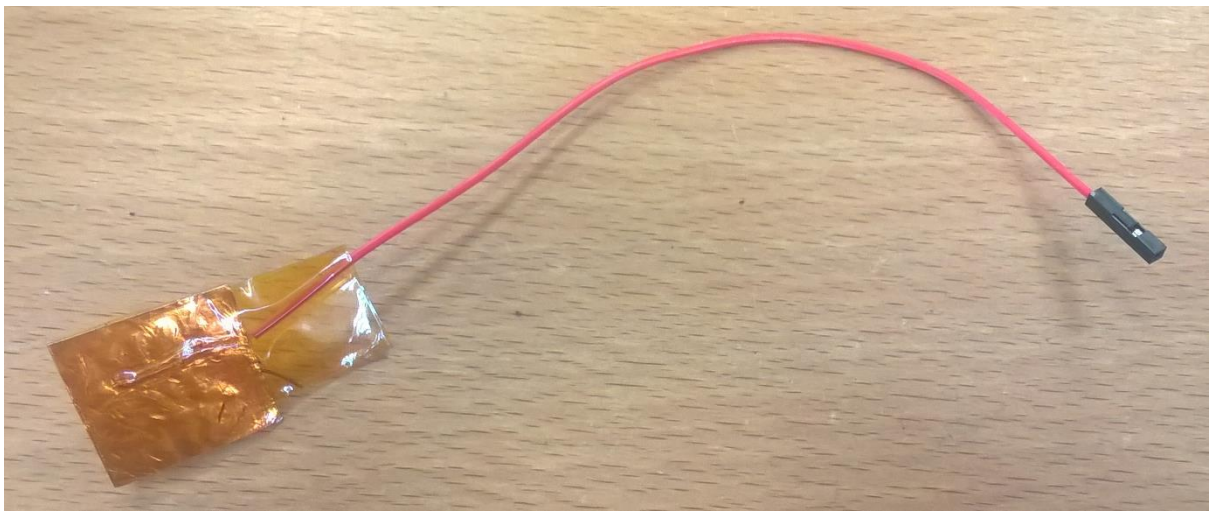


Figure 8: Kapton sensor

Figure 9: Measurement configuration for Kapton sensor

Results from Kapton Sensor

It was found that the results from the Kapton sensor were not repeatable or reproducible, and that there was no difference found above the noise between the measurements where rust was present and where there was no rust. It was also observed that the orientation of the rusty surface would make a difference to the reading, because the weight forcing the sensor and the surface together would have no effect if the rusty surface faced in any direction other than upwards. It was therefore considered that the way forward was to design a sensor that contained no dieiectric layer intrinsic to itself, and that another means of attracting the sensor to the surface other than gravity was needed. This would mean that the sensor would not return a signal if it were placed on rust-free metal because there would be no capacitor formed. The only time a signal would be present is when the sensor is placed on an area or rust thick enough to form a continuous dielectric layer across the area of the sensor.

Aluminium and neodymium sensor

The final sensor design took the following factors, learned from the shortcomings of the previous sensors in this project, into account:

1. No dielectric layer. The sensor had a solid, flat metallic interface that touched rust.
2. A force attracting the sensor to the rust surface. This replaces the weight used in the Kapton sensor described above.
3. Shielded, grounded cable connecting the sensor to the 555 timer circuit. It was found that the jumper leads shown in Figure 5, Figure 7 and Figure 8 acted as antennas and picked up mains hum at 50Hz, so the next sensor was designed to have grounded coaxial cable connecting sensor to circuit.

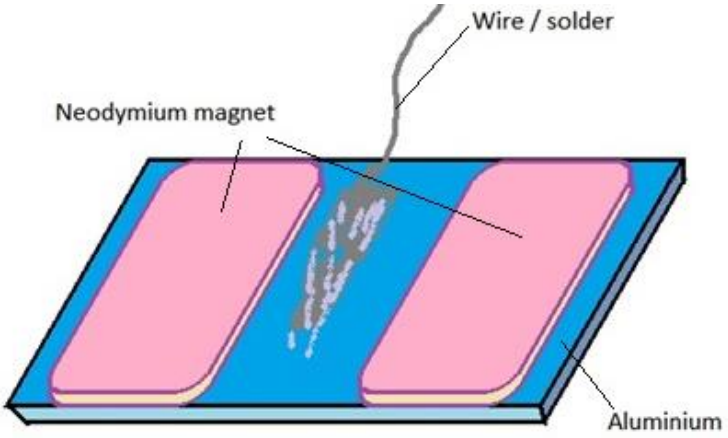


Figure 10: Aluminium and neodymium sensor

The magnets were used so that the sensor would attract to the steel underlying the rust layer, and the sensor was made from aluminium so that the main sensor material would not affect the strength of the magnetic field through it. The sensor was created and a photograph is shown in Figure 11. The connection lead from the circuit to the sensor was made from coaxial cable. The cable was shielded as it was found that when the sensor was connected via unshielded wire it acted as an antenna, picking up mains noise and other RF noise present in the electronics laboratory.

Figure 11: Practical realisation of Al and Nd sensor

The sensor shown in Figure 11 has an area of 45mm x 22mm or 0.00099m^2. The capacitance of this sensor, when it is placed on rust, can be calculated if the depth of the rust and its dielectric constant are known.

A value of 13.1 for the dielectric constant of rust was found from (5), and the thickness of the rust was found to be 0.21mm from ultrasonic NDT measurements.

The capacitance of the sensor when placed on this area of rust can be calculated as:

$$C_{rust} = \frac{\varepsilon_{rust}\varepsilon_0 A}{d}$$

$$= \frac{13.1 \times 8.854 \times 10^{-12} \times 0.00099}{0.21 \times 10^{-3}}$$

$$= 5.4680 \times 10^{-10} F$$

$$= 0.547\ nF$$

This capacitance value for the sensor in the 555 timer circuit shown in Figure 11 gives an output frequency of:

$$f = \frac{1}{\ln(2).C_1.(R_1 + 2R_2)}$$

$$= \frac{1}{0.693 \times 5.4680 \times 10^{-10} \times (3 \times 470 \times 10^3)}$$

$$= 1871.23\ Hz$$

Results with Al and Nd sensor

It was found that this sensor gave repeatable, reproducible results when used with the brake disc described below. The whole 555 timer circuit and sensor together are shown in Figure 12, which altogether can be viewed as the corrosion sensor.

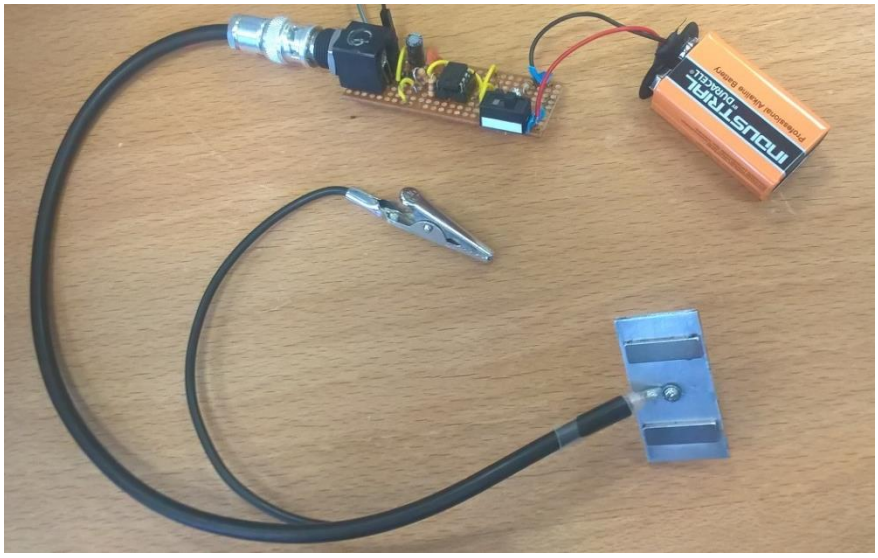


Figure 12: The corrosion sensor

A car brake disc was obtained that had different thicknesses of surface rust on its different surfaces. Three areas of interest were chosen and the corners of the sensor were marked as areas 1, 2 and 3. The top left spots showing the sensor placement areas can be seen in Figure 13. The sensor's ground clip was attached to the conductive outer rim of the disc and the sensor was placed on each of the three areas as seen in Figure 15. The output from pin 3 of the 555 timer was viewed on a Tektronix TDS2022C digital storage oscilloscope. As seen in Figure 16, the square wave frequency can be read from the oscilloscope screen. The results can be seen in the graph below and full results are shown in Table 1, Table 2 and Table 3.

Figure 13: Car brake disc showing areas 1, 2 and 3

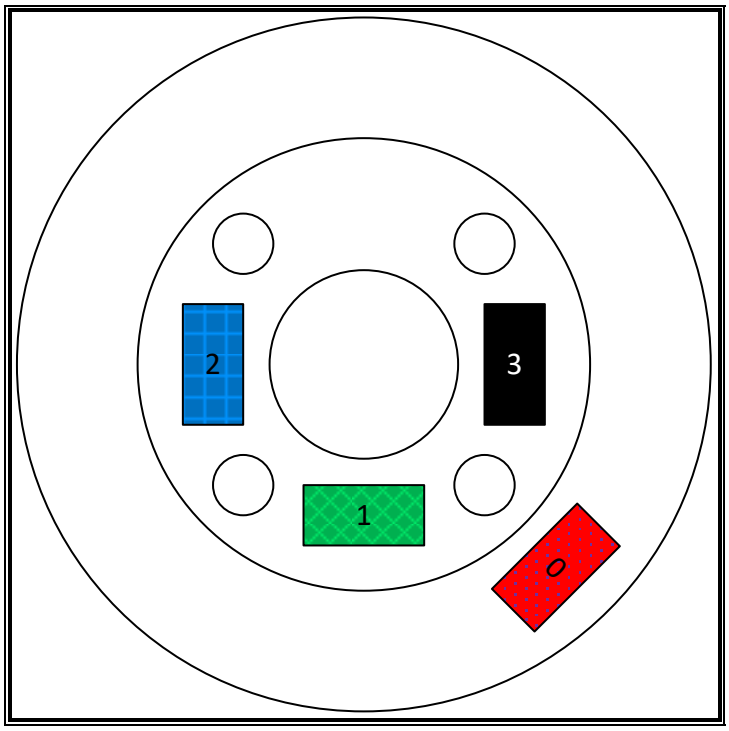


Figure 14: Sensor placement areas on brake disc

Figure 15: Brake disc showing sensor on area 1 and ground clip placement

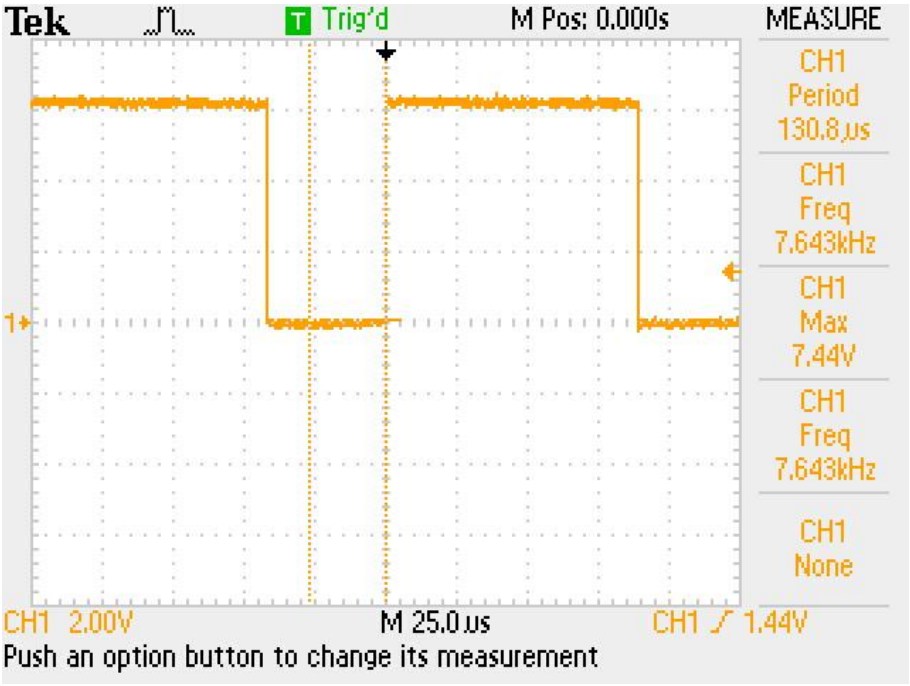


Figure 16: Output from corrosion sensor viewed on oscilloscope

Table 1: Results obtained from three areas of brake disc on day 1

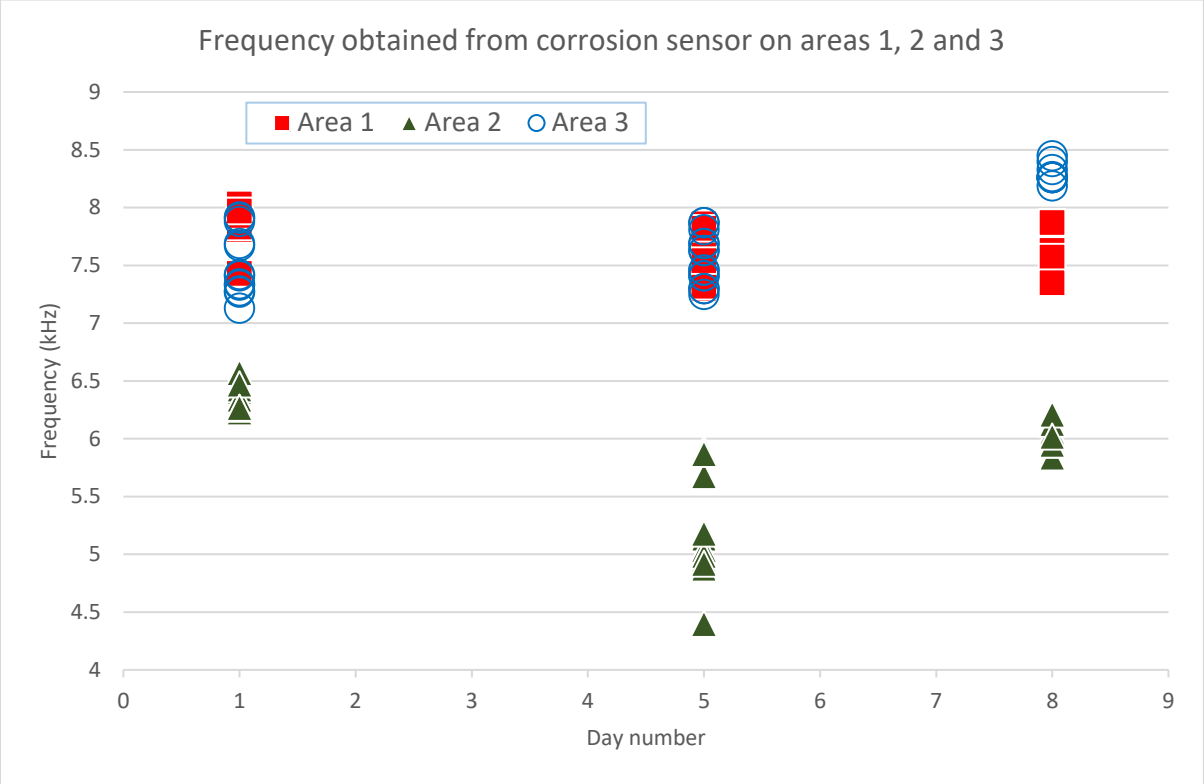
Frequency (kHz)			
	Area 1	Area 2	Area 3
	7.99	6.47	7.34
	7.85	6.28	7.69
	7.81	6.41	7.28
	7.81	6.57	7.42
	8.03	6.43	7.92
	7.97	6.34	7.91
	7.43	6.23	7.88
	7.83	6.47	7.13
	7.94	6.23	7.27
	7.94	6.27	7.67
	7.93	6.26	7.33
	7.97	6.27	7.41
Mean	7.875	6.3525	7.520833
Standard deviation (SD)	0.151630032	0.108714	0.267066
SD/Mean %	1.925460726	1.711359	3.551017
Capacitance calculated from mean	1.29956E-10	1.61E-10	1.36E-10

Table 2: Results obtained from three areas of brake disc on day 5

Frequency (kHz)			
	Area 1	Area 2	Area 3
	7.86	4.96	7.3
	7.74	5.14	7.3
	7.29	5.04	7.46
	7.75	4.88	7.69
	7.3	4.88	7.63
	7.51	5.2	7.46
	7.36	4.99	7.69
	7.61	4.4	7.25
	7.8	4.92	7.87
	7.45	5.68	7.41
	7.66	5.91	7.81
	7.32	5.88	7.63
	7.82	5.18	7.87
	7.54	5.87	7.41
Mean	7.572142857	5.209286	7.555714
Standard deviation (SD)	0.197707525	0.439081	0.206041
SD/Mean %	2.610985138	8.428819	2.726961
Capacitance calculated from mean	1.35154E-10	1.96E-10	1.35E-10

Table 3: Results obtained from three areas of brake disc on day 8

Frequency (kHz)			
	Area 1	Area 2	Area 3
	7.35	5.84	8.19
	7.52	5.95	8.26
	7.63	6.06	8.26
	7.35	6.13	8.26
	7.87	6.06	8.33
	7.57	6.13	8.45
	7.87	6.21	8.27
	7.87	5.95	8.26
	7.35	6.02	8.4
Mean	7.597777778	6.038889	8.297778
Standard deviation (SD)	0.214568185	0.106713	0.076562
SD/Mean %	2.824091355	1.767096	0.92268
Capacitance calculated from mean	1.34698E-10	1.69E-10	1.23E-10



Graph 1: Frequency obtained from corrosion sensor on areas 1, 2 and 3

Integration into wireless sensor system

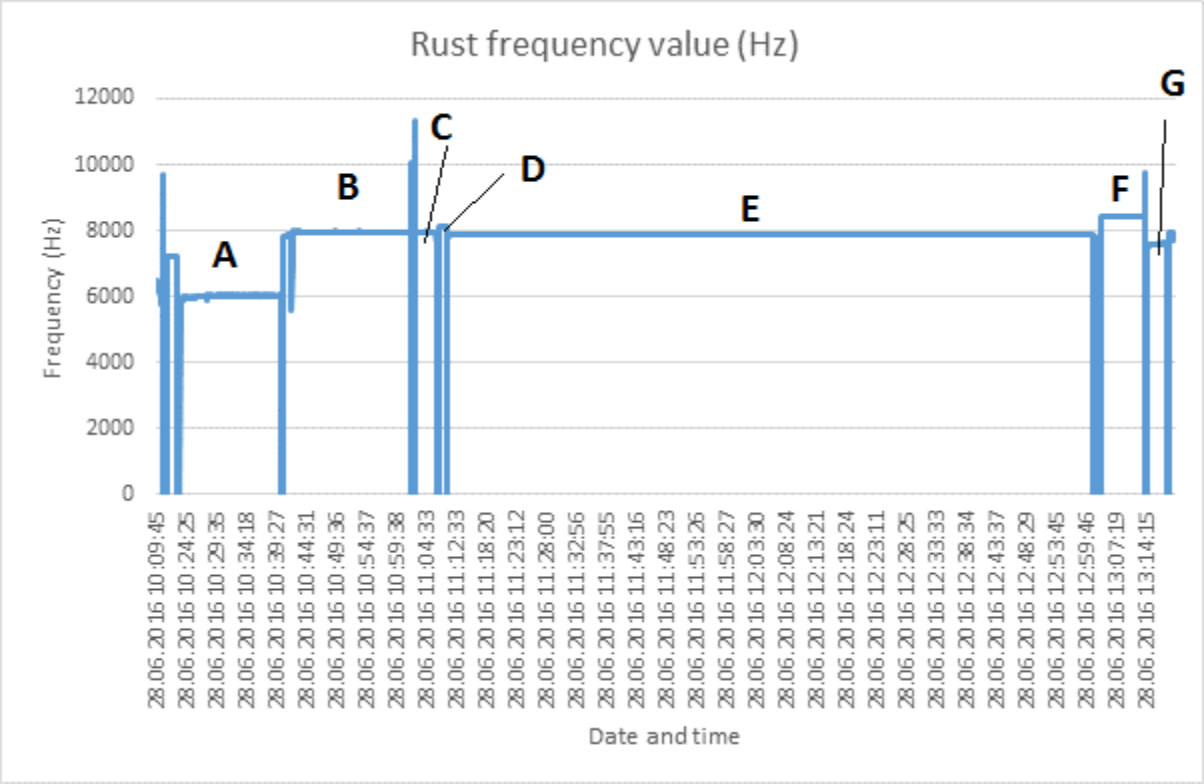
The signals from the corrosion sensor are suitable to be integrated into the wireless sensing system developed in the WIDESENSE project in 2015 (6). The data that needs to be transmitted is the frequency generated by the 555 timer circuit when the sensor detects rust.

Software obsolescence and updating issues

It was found that the WIDESENSE system could not be made to work again as it did in 2015. There were a few reasons for this. Firstly, the SD/RTC (Secure Datacard / Real Time Clock) shield library could not be located and Arduino had integrated their own SD and RTC libraries into subsequent versions of the IDE which were incompatible with the functions that were written last year. Once that was updated, a counter was implemented in the LilyPad Arduino code which counted, using the pulseIn() function, how many square wave rising edges were output by the corrosion sensor in the space of one second. This value was transmitted every second from the LilyPad and sent across the wireless link to the base station Arduino Mega, where it was recorded onto SD card with the RTC stamp giving time and date. A video was taken (7) showing the data from rust and from clean metal being transmitted.

Recorded Data

A comma separated values file was stored on the SD card over a four hour period, during which the sensor was occasionally moved to an area of no rust, and then moved back to the rust. The results can be seen below.



Graph 2: Frequency values obtained over wireless transmission

The letters A-G on the graph refer to different regions of results. Referring to Figure 14, regions A and G came from Area 2; regions B, C and E were from Area 1 and regions D and F were from Area 3.

It can be seen from the above graph that the frequency signal from one area is repeatable, within a certain amount of variation due to the exact placement of the sensor. A similar value is obtained when the sensor is replaced on the same area after a period on the rust-free shiny metal (area 0 in Figure 14) when the signal gave a value of 2Hz. The 2Hz signal was due to a bug in the software at the time when these measurements were made, and actually they refer to a signal of 0Hz or no capacitance. The Arduino code to transmit the data from the 555 timer circuit and to receive and store the data on the SD card can be found in the appendices.

Discussion of results

Sensor results

The frequencies obtained from all areas on each date was very different from that obtained by calculation. One reason for this could be that there is an air gap present between the layer of rust and the sensor. The rust is irregular, jagged and denticulate at microscopic level with some parts taller than others. The sensor is pushed up from the surface in places, allowing air to intrude and meaning that the capacitance of an irregular layer of air is being measured as well as the rust. It was found that the frequency seen on the oscilloscope reduced greatly when pressure was applied to the

sensor. This effect could be seen as pushing the air out of the way and reducing the capacitance of the air gap.

Looking at the difference between the theoretical capacitance of the area of rust covered by the sensor obtained by calculation in this report, and the mean capacitance for area 1 obtained on day 1 from experiment,

$$C_{experimental} = 1.29956 \times 10^{-10} F = 1.30 \times 10^{-10} F$$
$$C_{rust} = 5.4679 \times 10^{-10} F = 5.47 \times 10^{-10} F$$

$C_{experimental}$ is the total capacitance measured, including the capacitance of the rust and the capacitance of the air gap. These two capacitances can be thought of as being in series in an electronic circuit, so

$$C_{experimental} = \frac{C_{rust} \times C_{air}}{C_{rust} + C_{air}}$$

Solving for air gap capacitance C_{air} gives,

$$C_{air} = \frac{C_{experimental} \times C_{rust}}{C_{rust} - C_{experimental}}$$

Inserting the above values for $C_{experimental}$ and C_{rust},

$$C_{air} = \frac{1.30 \times 10^{-10} \times 5.47 \times 10^{-10}}{5.47 \times 10^{-10} - 1.30 \times 10^{-10}}$$

$$= 1.71 \times 10^{-10} F$$

In order to calculate what depth of air gap d this capacitance is produced by, the formula for capacitance is used with an ε_r of 1 and the area of the sensor $A = 45\text{mm x } 22\text{mm} = 0.00099\text{m}^2$:

$$C_{air} = \frac{\varepsilon_0 \varepsilon_r A}{d}$$

$$d = \frac{\varepsilon_0 A}{C_{air}}$$

$$= \frac{8.854 \times 10^{-12} \times 0.00099}{1.71 \times 10^{-10}}$$

$$= 5.13 \times 10^{-5} m \quad \text{or} \quad 51.3 \mu m$$

Limitations of technique

This technique is currently not waterproof, as the presence of water would act as a conductor and negate the dielectric effect of any rust present. It cannot in its present form be used to detect rust on underwater structures.

It has been found that shallow rust, and patchy rust, do not give a signal. This is because the sensor covers an area that includes a part where bare metal shows through, touching the sensor, thus meaning that no capacitance effect is present.

This technique could be refined with further work to be used to sense the thickness of a layer of rust. At present, however, the readings obtained merely give an indication as to the presence or absence of a rust layer.

Wireless Transmission

Wireless transmission within the system created for the WIDESENSE project was found to work correctly, opening the way for future work in creating a wirelessly distributed sensor array. At the moment, one rust sensor's data is successfully transmitted to a base station that receives and stores the data. A wirelessly distributed sensor array would collate data from several sensors to one base station, and possibly also do some data analysis to check whether, if multiple sensors reported corrosion, the area needed attention from corrosion engineers. This is shown in the diagrams below.

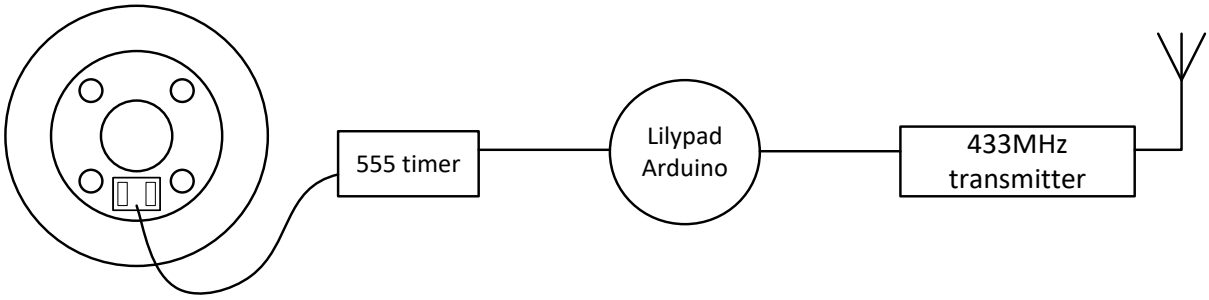


Figure 17: Block diagram of sensor transmission – WIDESENSE configuration

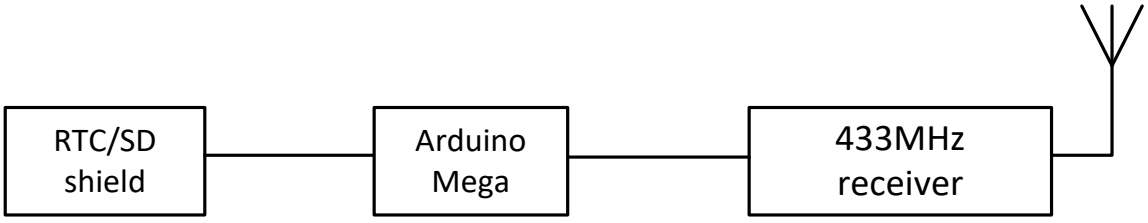


Figure 18: Block diagram of sensor data reception and storage on SD - WIDESENSE configuration

In a wirelessly distributed array, each sensor would have the electronic circuitry of the 555 timer and the radio transmitter miniaturised into a System-on-Chip (SoC) configuration that would be attached to each sensor. The sensors could then be placed across a wide surface and could all transmit data to a single base station that then sends the data on to a laptop computer where it is analysed either

by a human operator or by data processing to determine whether there is a corrosion issue that needs intervention.

In this scenario it is envisaged that each sensor would appear as in Figure 19, and the whole wirelessly distributed array, for example measuring a large area such as a ship's funnel, would appear as in Figure 20.

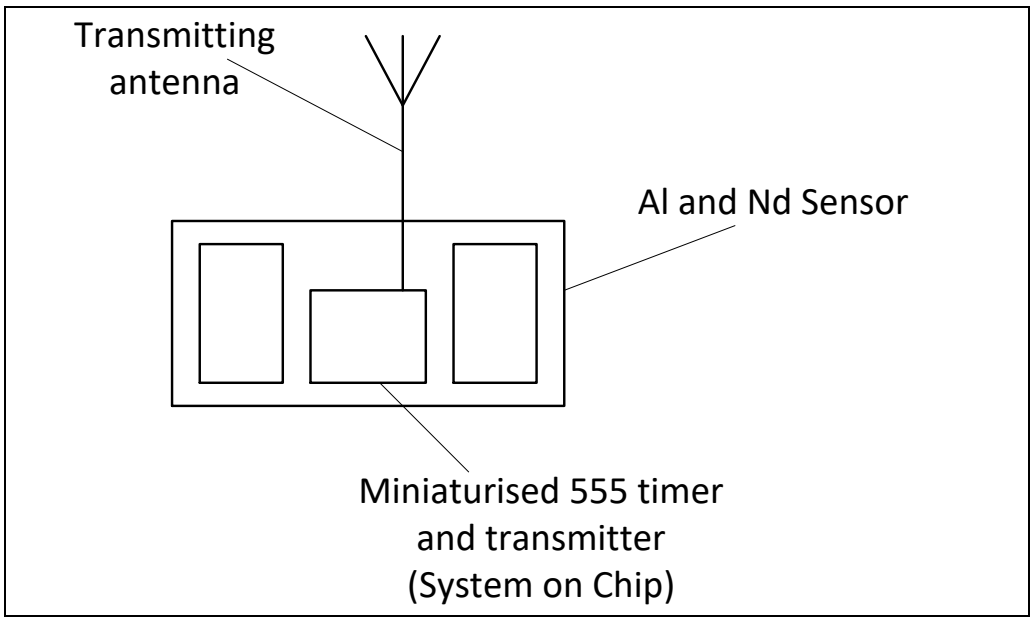


Figure 19: Self-contained transmitting sensor

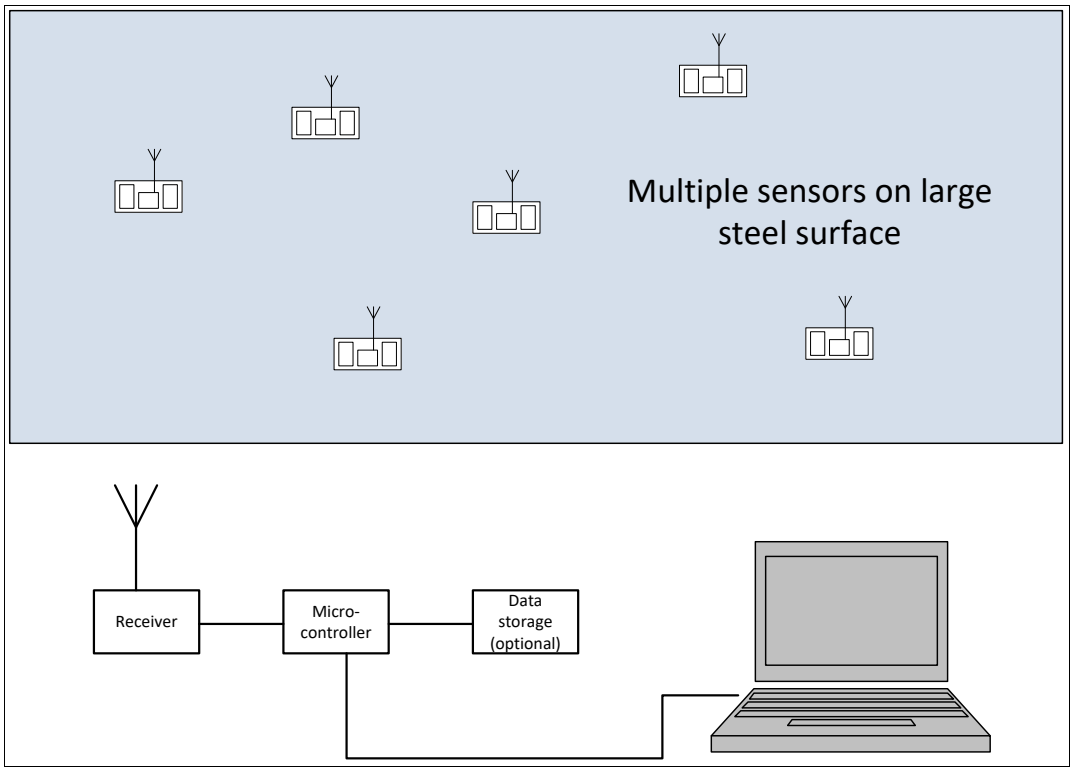


Figure 20: Autonomous corrosion sensing using wireless distributed array

It was found that data could only be reliably received if the transmission end was correctly set up and the ground connection was continuous throughout the sensor, 555 timer circuit, microprocessor and transmitter. It is envisaged that when this system is further developed, all of these elements will be co-located within a very small volume, and sharing the ground connection will be more intuitive and require less effort than it did in this project's setup.

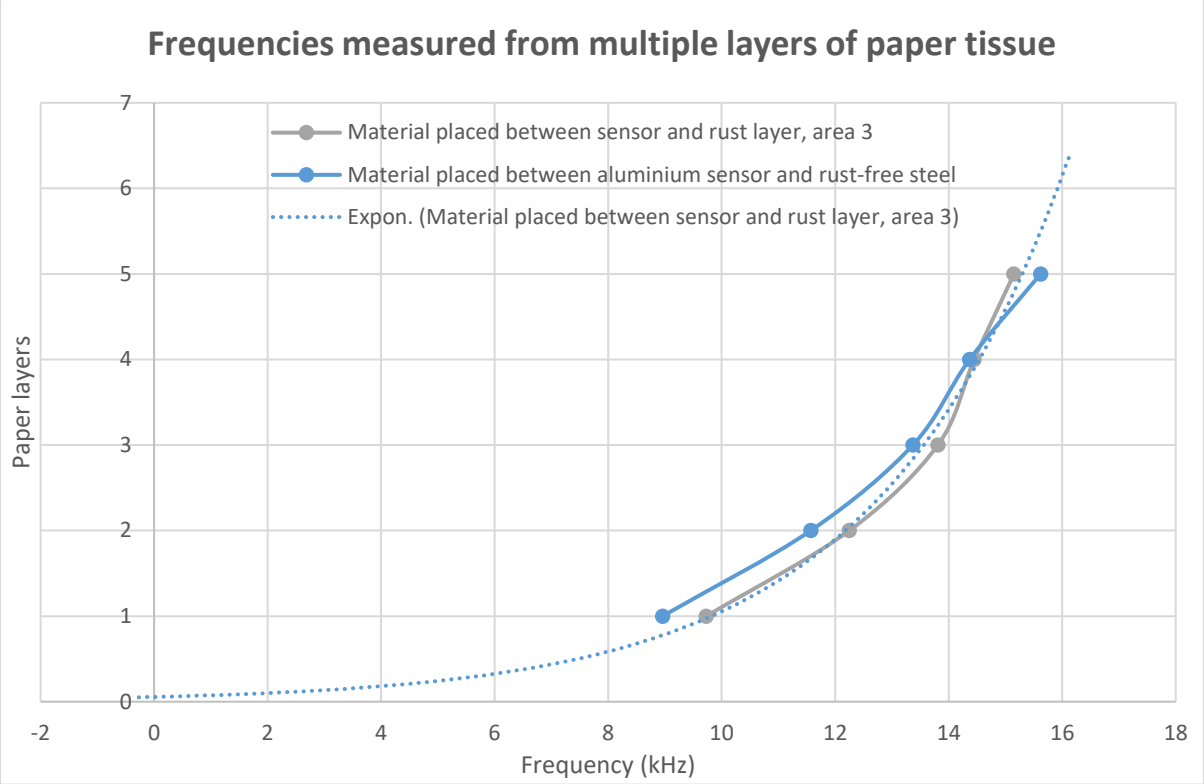
Other work using corrosion sensor

A range of other materials were placed between the sensor plate and the rust-free rim of the brake disc, and between the sensor and the rusty area 3. Layers of tissue paper were measured and the results can be seen below.

Table 4: Frequency measurements from multiple layers of paper tissue

Frequencies obtained from 555 timer circuit (kHz)			
Material placed between sensor and rust layer, area 3		**Material placed between aluminium sensor and rust-free steel**	
Material	**Frequency (kHz)**	**Material**	**Frequency (kHz)**
Paper tissue, 1 layer	9.73	Paper tissue, 1 layer	8.96
Paper tissue, 2 layers	12.25	Paper tissue, 2 layers	11.57
Paper tissue, 3 layers	13.81	Paper tissue, 3 layers	13.37
Paper tissue, 4 layers	14.45	Paper tissue, 4 layers	14.37
Paper tissue, 5 layers	15.15	Paper tissue, 5 layers	15.62

A graph was created with the above measurements and a trendline (exponential) was overlaid for the data obtained without rust.



Graph 3: Frequencies obtained from multiple layers of paper tissue

It can be seen from the trendline that when the data is extrapolated to 0 layers of paper tissue, the origin is close to being crossed. This would be logical as when there are no layers of paper between the sensor and the metal, the capacitance is zero and therefore the frequency obtained is zero.

Conclusions and Further Work

A system has been implemented that uses the dielectric properties of rust deposits to measure the presence and qualitative thickness of a corrosion layer on a steel brake disc. Further investigation is needed to quantify the depth of rust that can be detected using this method, and how it relates to the structural integrity of the remaining structure.

If the electronic circuitry could be minimised such that it could be integrated onto the back of the sensor, then an array of such sensors across a steel structure could be used to create a wireless array that would monitor the health of the steel effectively and over a long time period. This may pave the way for the sensor, in miniature form, to become waterproofed and therefore be capable of being used in the splash or intertidal zones.

Other corrosion manifestations could be detected using the capacitance method, for example, crevice and intergranular corrosion. In these cases, corrosion forms along discontinuities in the metal, forming a capacitive structure within the metal rather than on the surface. The sensing points either side of the dielectric would have to be redesigned, but the same method could be utilised.

This technique could also be used to detect other inclusions on the metal, for example corrosion under a paint layer, which would otherwise be missed. Further work is necessary into the capacitance of the paint layer and how the reading changes when corrosion is present.

Wireless transmission of data

Although a multi-sensor wireless array has not been achieved, it has been shown that data from a single sensor can be reliably transmitted, received and stored over a long time period. Further work is required to make a wirelessly distributed array work with more than one sensor transmitting to a single base station.

It is also hoped that future work will include minimisation of the corrosion sensor such that the 555 timer, microprocessor and transmitter can all be fitted onto the top of the aluminium plate. Potential issues with doing this include the presence of the neodymium magnets and how these would interfere with the functioning of the microprocessor and the transmitter. However, if these potential problems could be overcome, it might be possible to encase the whole sensor in a waterproof housing (allowing the metal to interface with the surface that it is measuring) and the sensor could be used below the water surface.

Acknowledgements

The author would like to extend heartfelt thanks to Frank Noakes, without whose superior technician skill this project would have nosedived.

Also, the author would very much like to thank Mr A. Twigg and SJP Vehicle Servicing Repairs and MOTs (www.sjpserviceandmot.co.uk) for providing the brake disc, and Kevin Thatcher for help in trying to make things rust and using the XRF gun.

References

1. **Various.** Capacitor. *Wikipedia.* [Online] May 29, 2016. [Cited: June 9th, 2016.] https://en.wikipedia.org/wiki/Capacitor.

2. Electrical Properties of Materials. [Online], 1983. [Cited: 06 27, 2016.] http://www.drexelbrook.com/download/420-001-569-pdf.pdf.

3. Standard corrosion protection systems for buildings. [Online] 2013. [Cited: 06 27, 2016.] http://www.steelconstruction.info/Standard_corrosion_protection_systems_for_buildings.

4. **Dr. Edward Sayre, Mr. Michael Baxter, Dr. Jinhua Chen.** Limits of FR-4 in High Speed Designs. [Online] [Cited: June 10, 2016.] http://www.ieee802.org/3/10G_study/public/jan00/sayre_1_0100.pdf.

5. **AMETEK Ltd.** Electrical Properties of Materials. *www.drexelbrook.com.* [Online] [Cited: July 5, 2016.] www.drexelbrook.com/download/420-001-569-pdf.pdf.

6. Twigg, Helena and Molinari, Marc. (2015). *WIDESENSE - Wireless & Distributed Sensor Enhancement - Project report*. Project Report. Southampton Solent University. http://ssudl.solent.ac.uk/3235/

7. **Twigg, H.K.** *Video of corrosion sensor working over WIDESENSE wireless system.* Southampton : s.n., 2016.

8. 555 timer IC. [Online] 2016. [Cited: June 9th, 2016.] https://en.wikipedia.org/w/index.php?title=555_timer_IC&oldid=721856248.

Appendix 1 – LilyPad Arduino transmission code

```
/* Code to transmit from Lilypad Arduino
 *  Modified for addition of Corrosion Sensor
 *  240616
 *  Everything else removed except for corrosion sensor data
 *  so that frequency count might be correct
 *  240616 12:52
 *  Measure pulseIn up time and down time and determine frequency from these.
*  Helena Twigg
 */

#include <math.h>
#define rfTransmitPin 4  //RF Transmitter pin = digital pin 4
#define ledPin 13        //Onboard LED = digital pin 13



int corrPin = A2;

const long interval = 1000;  //There are 1000 milliseconds in a second
unsigned long prevMillis =0;   // Used for counting to one second
unsigned long currentMillis;  // Used for counting to one second

int corrFrequency;  //variables for sensors



void setup() {
  pinMode(rfTransmitPin, OUTPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  pinMode(corrPin, INPUT);
}
```

```
void SendHIGH()
{
  //Serial.print("H");
  digitalWrite(rfTransmitPin, HIGH);    //Transmit a HIGH signal
  digitalWrite(ledPin, HIGH);          //Turn the LED on
  delay(40);                          //timings are weird because of the FTDI or lilypad or something, so
actual value is half
  digitalWrite(rfTransmitPin, LOW);    //Transmit a LOW signal
  digitalWrite(ledPin, LOW);          //Turn the LED off
  delay(20);
}
void SendLOW()
{
  //Serial.print("L");
  digitalWrite(rfTransmitPin, HIGH);    //Transmit a HIGH signal
  digitalWrite(ledPin, HIGH);          //Turn the LED on
  delay(20);                          //timings are weird because of the FTDI or lilypad or something, so actual
value is half
  digitalWrite(rfTransmitPin, LOW);    //Transmit a LOW signal
  digitalWrite(ledPin, LOW);          //Turn the LED off
  delay(40);
}

void sendStart()
{
  // send 40ms LONG signal
  digitalWrite(rfTransmitPin, HIGH);    //Transmit a HIGH signal
  digitalWrite(ledPin, HIGH);          //Turn the LED on
  delay(80);                          //timings are weird because of the FTDI or lilypad or something, so
actual value is half
  digitalWrite(rfTransmitPin, LOW);    //Transmit a LOW signal
  digitalWrite(ledPin, LOW);          //Turn the LED off
  delay(20);
```

```
}
void sendEnd()
{
  for (int i = 0; i < 8; i++)
  {
    SendHIGH();
  }
  for (int i = 0; i < 8; i++)
  {
    SendLOW();
  }
}

void sendByte(char c)
{
  char t = c;
  //Serial.print("binary being sent: " );
  for (int i = 0; i < 8; i++)
  {
    // send highest bit
    if (t & 128) {
      SendHIGH();
    }
    else
    {
      SendLOW();
     }

    // shift right
    t = t << 1;
  }
```

```
}

void sendValue(int v)
{
  int a = v;
  for (int i = 0; i < 16; i++)
  {
    if (a & 32768) {
      SendHIGH();
    }
    else
    {
      SendLOW();
    }
    //shift right
    a = a << 1;
  }
}

void sendSensor(char sensorID)
{
  char ch = sensorID;
  for (int i = 0; i < 8; i++)
  {
    if (ch & 128) {
      SendHIGH();
    }
    else
    {
      SendLOW();
    }
    //shift right
```

```
    ch = ch << 1;
  }
}

int Thermistor(int RawADC) {
  //Serial.print("Temp RawADC = ");
  //Serial.print(RawADC);
  //Serial.print("    ");
  double Temp;
  Temp = log(10000.0*((1024.0/RawADC-1)));
  Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp ))* Temp );
  Temp = Temp - 273.15;            // Convert Kelvin to Celsius
   //Temp = (Temp * 9.0)/ 5.0 + 32.0; // Convert Celsius to Fahrenheit
   return int(Temp)  ;
}



void loop() {

  corrFrequency = 1; //What's the frequency from the 555 timer?

  //prevMillis; //Need to count to 1000 for each second

  prevMillis = millis();
  currentMillis = millis();

  //need to count for a second and then send values.
  while((currentMillis - prevMillis) < interval)
  {
    //unsigned long duration = pulseIn(corrPin, HIGH);
    if(pulseIn(corrPin, HIGH)){
    corrFrequency++;
```

```
  }
  currentMillis = millis();
 }
 // For the Lilypad must multiply frequency by two due to timing issue
 corrFrequency = corrFrequency*2;
 prevMillis = currentMillis;

 sendStart();
 sendByte('r');  //rust sensor
 Serial.print("Rust data is ");
 Serial.println(corrFrequency);
 sendValue(corrFrequency);
 sendEnd();

}
```

Appendix 2 – Arduino Mega code to receive and store sensor data

```
/* Arduino wireless receive for an Arduino MEGA
 *  Modified 210616 addition of rust sensor
 *  Modified 240616 removal of SD
 *  Modified 240616 readdition of different SD
 *  Modified 270616 Addition of different RTC
 *  Modified 280616 transmitter "noise"

   Written and designed by Helena Twigg

   280616
*/
#include <SPI.h>
#include <SD.h>
#include <DS1307.h>

#define rfReceivePin A0  //RF Receiver pin = A0
#define ledPin 13        //Onboard LED = digital pin 13

//from carAccelerator
bool carderror = false;

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;
File dataFile;
String strFile;
char filename[100] = "0";

// Change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
```

```
// Sparkfun SD shield: pin 8
const int chipSelect = 4;

//////////////////end of from carAccelerator

//variable to hold SD chip select pin - updated SD.h??
// commented out for now #define chipSelect 5

//variables to store received data
unsigned long HIGHDuration = 0;  // length of received pulse
unsigned int receiveArray = 0;  //Array in which to store received bits
char sensorType;
unsigned int numberOfSensors = 0;
unsigned int sensorsRemaining = 0;
int sensorData;
static int receivedBits = 0;  //counter of received bits
//File dataFile;
// make a string for assembling the data to log:
String dataString = "";

//state flags
boolean messageStarted = false;
boolean messageEnded = false;
boolean sensorChosen = false;
boolean dataReceived = false;
boolean knowHowManySensors = false;

// RTC settings
// Init the DS1307 with:
// DS1307:  SDA pin   -> Arduino port 4 now 20
//          SCL pin   -> Arduino port 5 now 21
DS1307 rtc(20, 21);
```

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(rfReceivePin, INPUT);
  Serial.begin(19200);
  Serial.println("We've got to setup");
  //if (!SD.begin(10, 11, 12, 13))   //This used to work
/*  if(!SD.begin(chipSelect))      //will make it compile, possibly wrong - updated SD.h??
  {
    //for(;;)
    {
      //Serial.println("No SD Card, help...");
      // Set up real time clock - RTC
      // Set the clock to run-mode
     /* rtc.halt(false);
      Serial.println(rtc.getDateStr());
      Serial.println(rtc.getTimeStr());
    }
    //while(1); //Wait forever because no SD Card accessible
  }*/

  // Set up real time clock - RTC
  rtc.begin();
  // Set the clock to run-mode
  rtc.halt(false);
  Serial.println(rtc.getDateStr());
  Serial.println(rtc.getTimeStr());

  // The folloMwing lines can be commented out to use the values already stored in the DS1307
//  rtc.setDOW(MONDAY);        // Set Day-of-Week to SUNDAY
//  rtc.setTime(15,25,00);     // Set the time to 12:00:00 (24hr format)
```

```
//  rtc.setDate(27, 06, 2016);   // Set the date to October 3th, 2010


  // Open up the file we're going to log to and name it with a date
  /*dataFile = SD.open("sensors.csv", FILE_WRITE);
  if (!dataFile) {
    Serial.println( ("error opening sensors.csv"));
    // Wait forever since we cant write data
    while (1) ;
  }
  //Start the data string that will eventually be our .csv file
  dataFile.println("Time, Temperature, Light, Rust");*/


//from carAccelerator
// Setup SD card
   Serial.print("\nInitializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by default.
  // Note that even if it's not used as the CS pin, the hardware SS pin
  // (10 on most Arduino boards, 53 on the Mega) must be left as an output
  // or the SD library functions will not work.
  pinMode(10, OUTPUT);     // change this to 53 on a MEGA, didn't work so assumed MEGA is
extended UNO


  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  pinMode(chipSelect, OUTPUT);
  digitalWrite(chipSelect, HIGH);
  delay(200);
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card is inserted?");
    Serial.println("* Is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
```

```
  carderror = true;
}
else
{
  Serial.println("Wiring is correct and a card is present.");

  // print the type of card
  Serial.print("\nCard type: ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }
  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");
    carderror = true;
    //lcd.clear();
    //lcd.print("SD FAT16/32 error.");
    //return;
  }
}
if (!carderror) {
  // print the type and size of the first FAT-type volume
```

```
  uint32_t volumesize;
  Serial.print("\nVolume type is FAT");
  Serial.println(volume.fatType(), DEC);
  Serial.println();

  volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
  volumesize *= volume.clusterCount();       // we'll have a lot of clusters
  volumesize *= 512;                         // SD card blocks are always 512 bytes
  Serial.print("Volume size (bytes): ");
  Serial.println(volumesize);
  Serial.print("Volume size (Kbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Mbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);

  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);
}

// create file to store sensor data
// see if the card is present and can be initialized:
if (!SD.begin(chipSelect)) {
  Serial.println("Card failed, or not present.");
  carderror = true;
}
else
{
```

```
   Serial.println("Card initialized.");
   // open the file. note that only one file can be open at a time,
   // so you have to close this one before opening another.

   /*int i=0;
   do
   {
    i = i+1;
    strFile = "data_" + String(i) + ".csv";
    strFile.toCharArray(filename, strFile.length()+1);
   } while( SD.exists(filename) );

   Serial.println("====================================");
   Serial.print("  filename = ");
   Serial.println( filename );
   Serial.println("====================================");*/
  }
//////////////end of from carAccelerator

// Open up the file we're going to log to and name it with a date
  dataFile = SD.open("sensors.csv", FILE_WRITE);
  if (!dataFile) {
   Serial.println( ("error opening sensors.csv"));
   // Wait forever since we cant write data
   while (1) ;
  }
  //Start the data string that will eventually be our .csv file
  //Unfortunately not got rtc working so can't create timestamped data
  //dataFile.println("Time, Temperature, Light, Rust");
  dataFile.println("Time, Rust frequency value (Hz)");

}
```

```
void loop() {
  HIGHDuration = pulseIn(rfReceivePin, HIGH);  //returns length (in time) of pulse in microseconds, or 0 if it waits more than a second
  //Serial.println("We've got to the loop");
  boolean received = decodePulse();  //find out if it is a 1 or a 0
  if (received)
  {
    checkState();
  }
}

boolean decodePulse()
{
  if ((HIGHDuration > 8000) && (HIGHDuration < 15000))  //short pulse is a 0
  {
    receiveArray = receiveArray << 1;  //shift array along one bit
    bitWrite(receiveArray, 0 , 0);  //wite a low bit to the lowest bit of the receiveArray
    receivedBits++;    //increment receivedBits counter
    Serial.print("0");
    return true;
  }
  else if ((HIGHDuration > 15000) && (HIGHDuration < 25000))  //long pulse is a 1
  {
    receiveArray = receiveArray << 1;  //shift array along one bit
    bitWrite(receiveArray, 0, 1);   //write a high bit to the lowest bit of the receiveArray
    receivedBits++;    //increment receivedBits counter
    Serial.print("1");
    return true;
  }
  else if ((HIGHDuration > 25000)&&(HIGHDuration < 45000))
```

```
  {
    messageStarted = true;
    receivedBits = 0;
    Serial.print("Message Started");
    Serial.println(messageStarted);
    knowHowManySensors = false;
    messageEnded = false;
    sensorChosen = false;
    dataReceived = false;
    //dataString = String(rtc.getDateStr()) + " " + String(rtc.getTimeStr()) + ",";
    dataString = "";
    return true;
  }
  else if ((HIGHDuration > 0)&&(HIGHDuration < 8000))
  {
    return false;
  }
  //Serial.print("HIGHDuration =");
  //Serial.println(HIGHDuration);
  Serial.print("X");
  return false;


}

void checkState()
{
  if (receivedBits >= 16)
  {
    if (messageStarted == false)
    {
      //checkMessageStart();
```

```
    return;
  }
}

//else do nothing, just keep receiving bits
if (messageStarted == true)
{
  //Serial.print("knowHowManySensors: ");
  //Serial.println(knowHowManySensors);
  if (knowHowManySensors == false)
  {
    if (receivedBits == 16)
    {
      //Serial.print("receivedBits: ");
      //Serial.println(receivedBits);
      //find out how many sensors' data are going to be in this message
      howManySensors();
      Serial.print("Number of sensors in this message is ");
      Serial.println(numberOfSensors);

      //dataString += String(numberOfSensors)+ ",";
      return;
    }
  }
}

if (sensorsRemaining)
{
  if (!sensorChosen)
  {
    if (receivedBits == 8)
    {
```

```
      whichSensor();
      return;
     }
    } else if (sensorChosen) //or just else, but clarity needed for now
    {
     if (receivedBits == 16)
     {
      sensorsRemaining--;
      //Write date and time to datastring, crap place but there's nowhere better
      dataString = String(rtc.getDateStr()) + " " + String(rtc.getTimeStr()) + ",";
      receiveByte(); //which is now two bytes
      sensorChosen = false; //for the next sensor
      return;
     }
    }
  }
  if (dataReceived)
  {
    if (receivedBits == 16)
    {
     checkMessageEnd();
    }
  }
}



void checkMessageEnd()
{
  if (receiveArray == 65280)  //1111111100000000 in binary, sent as end of message
  {
    messageEnded = true;
```

```
   //reset all flags
   messageStarted = false;
   knowHowManySensors = false;
   sensorChosen = false;
   dataReceived = false;
   receivedBits = 0;
   Serial.println("Message Ended");

   dataFile.println(dataString);// nothing in datastring so:
   // The following line will 'save' the file to the SD card after every
   // line of data - this will use more power and slow down how much data
   // you can read but it's safer!
   // If you want to speed up the system, remove the call to flush() and it
   // will save the file only every 512 bytes - every time a sector on the
   // SD card is filled with data.
   dataFile.flush();
   Serial.print(dataString);
   Serial.println();


  }

}


void whichSensor()
{
  //  read the lower byte of the receive array
  sensorType = char(lowByte(receiveArray));
  //  For now, just print out what letter it is
  Serial.print("Sensor received is ");
  //dataString += String(sensorType) + ",";
```

```
  switch (sensorType) {
    case 'l':
      Serial.print("light");
      Serial.println();
      break;
    case 't':
      Serial.print("temperature");
      Serial.println();
      break;
    case 'r':
      Serial.print("rust");
      Serial.println();
      break;
    default:
      Serial.print("Sensor Type unknown: ");
      Serial.println(sensorType);
  }

  //set the flag for the next state
  sensorChosen = true;
  //reset receivedBits
  receivedBits = 0;
}

void howManySensors()
{
  //  read the lower byte of the receive array
  numberOfSensors = receiveArray;
  sensorsRemaining = numberOfSensors;
  //set the flag for the next state
  knowHowManySensors = true;
  //reset receivedBits
```

```
  receivedBits = 0;
}

void receiveByte()
{
  sensorData = receiveArray;
  // Decide what to do with it based on the sensorType
  switch (sensorType) {
    case 'l':
      //do something useful with the light data
      Serial.print("and the light data is equal to ");
      Serial.print(sensorData);
      Serial.println();
      //dataString += String(sensorData) + ",";
      break;
    case 't':
      //do something meaningful with the temperature data
      Serial.print("and the temperature data is equal to ");
      Serial.print(sensorData);
      Serial.println();
      //dataString += String(sensorData) + ",";
      break;
    case 'r':
      //do something meaningful with the rust data
      Serial.print("and the rust data is equal to ");
      Serial.print(sensorData);
      Serial.println();
      dataString += String(sensorData);
  }
  dataReceived = true;
  Serial.print("sensorsRemaining = ");
  Serial.println(sensorsRemaining);
```

```
  if (sensorsRemaining)
  {
    sensorChosen = false;
    dataReceived = false;
  }
  receivedBits = 0;
}
```